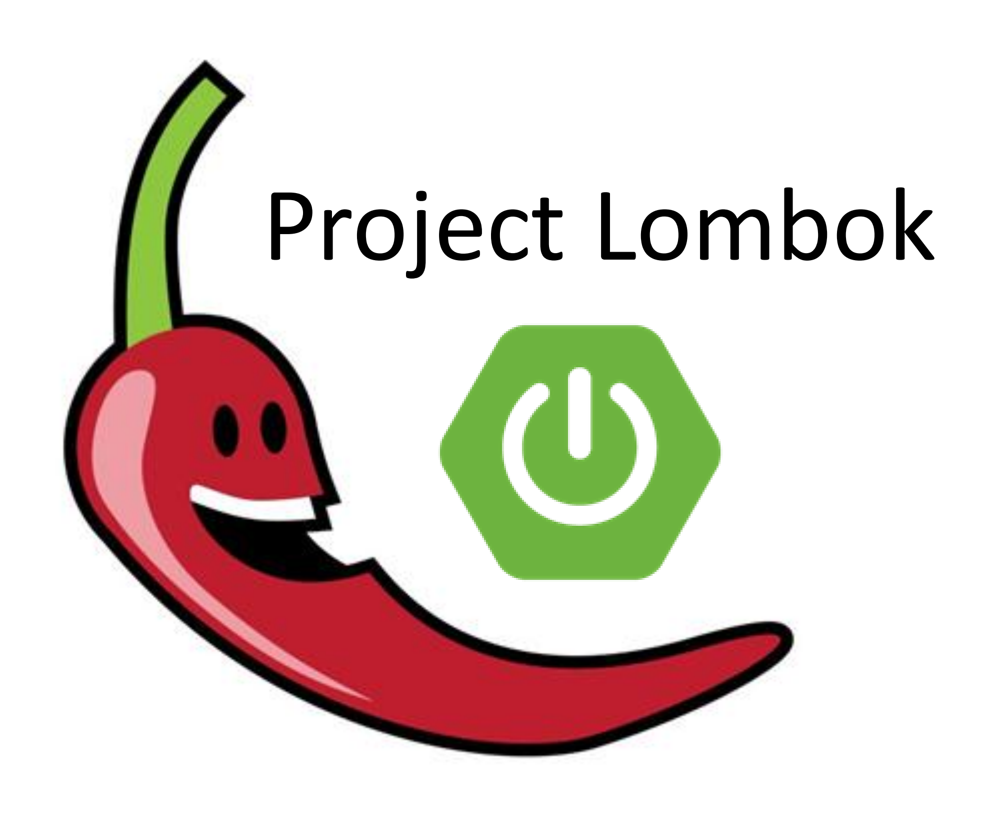


LOMBOK AVEC SPRING BOOT



PLAN DU COURS

– Introduction

– Fonctionnement Lombok

– Installation

– Les annotations du projet Lombok

– Place au pratique

INTRODUCTION

- Le projet Lombok est une bibliothèque Java qui **génère du code** en respectant de nombreuses bonnes pratiques.
- Cette bibliothèque remplace le code par des annotations faciles à utiliser ce qu'on appelle du « **boiler plate** » :

  ✔Getters, Setters et Constructeurs (vides et avec paramètres)

  ✔Equals et HashCode

  ✔To String

  ✔Modificateurs d'accès (private, protected, etc.)

  ✔Logger

FONCTIONNEMENT LOMBOK

- Lombok est un processeur d'annotation, qui **fonctionne au moment de la compilation** pour ajouter des codes dans les classes.
- Lorsque on utilise cette bibliothèque, pendant la compilation, le compilateur donne au processeur d'annotation Lombok la possibilité d'ajouter tout ce qui doit être ajouté à l'intérieur de la classe. Donc il ajoute ce qui suit dans la classe.

INSTALLATION – DÉPENDANCE

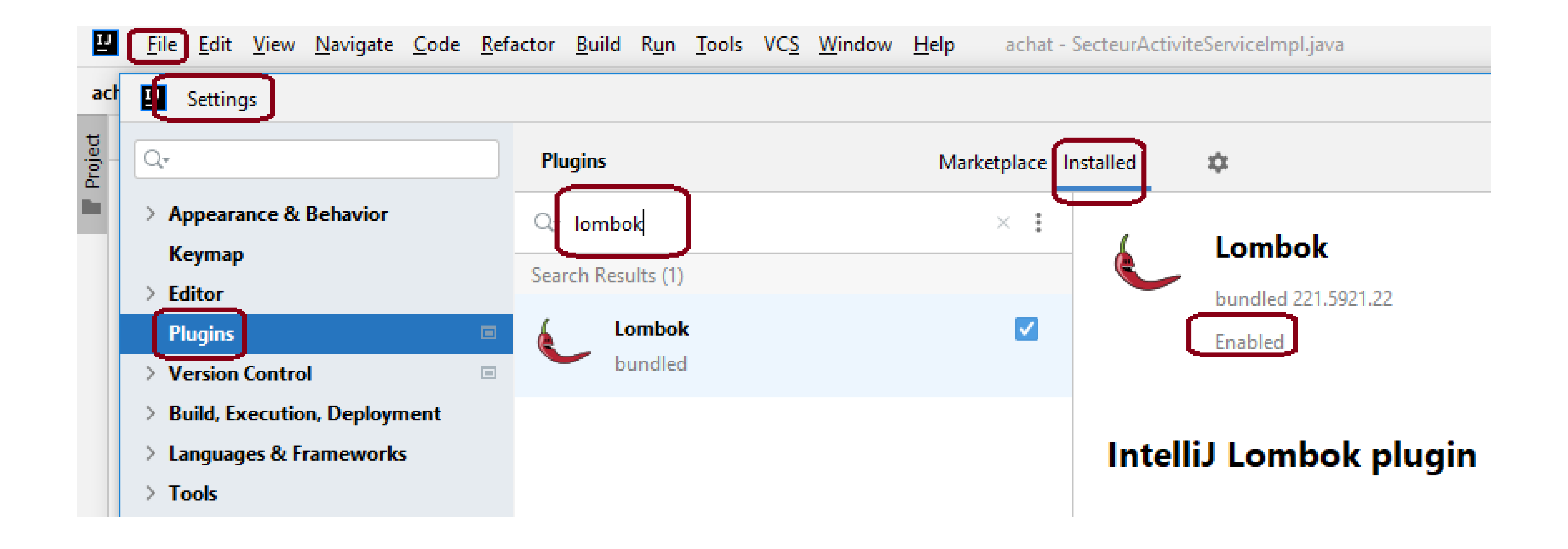
- Pour intégrer Lombok dans un projet Spring Boot, il suffit juste d'importer la dépendance dans le fichier **POM.xml**:

```
<dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
</dependency>
```

INSTALLATION - PLUGIN

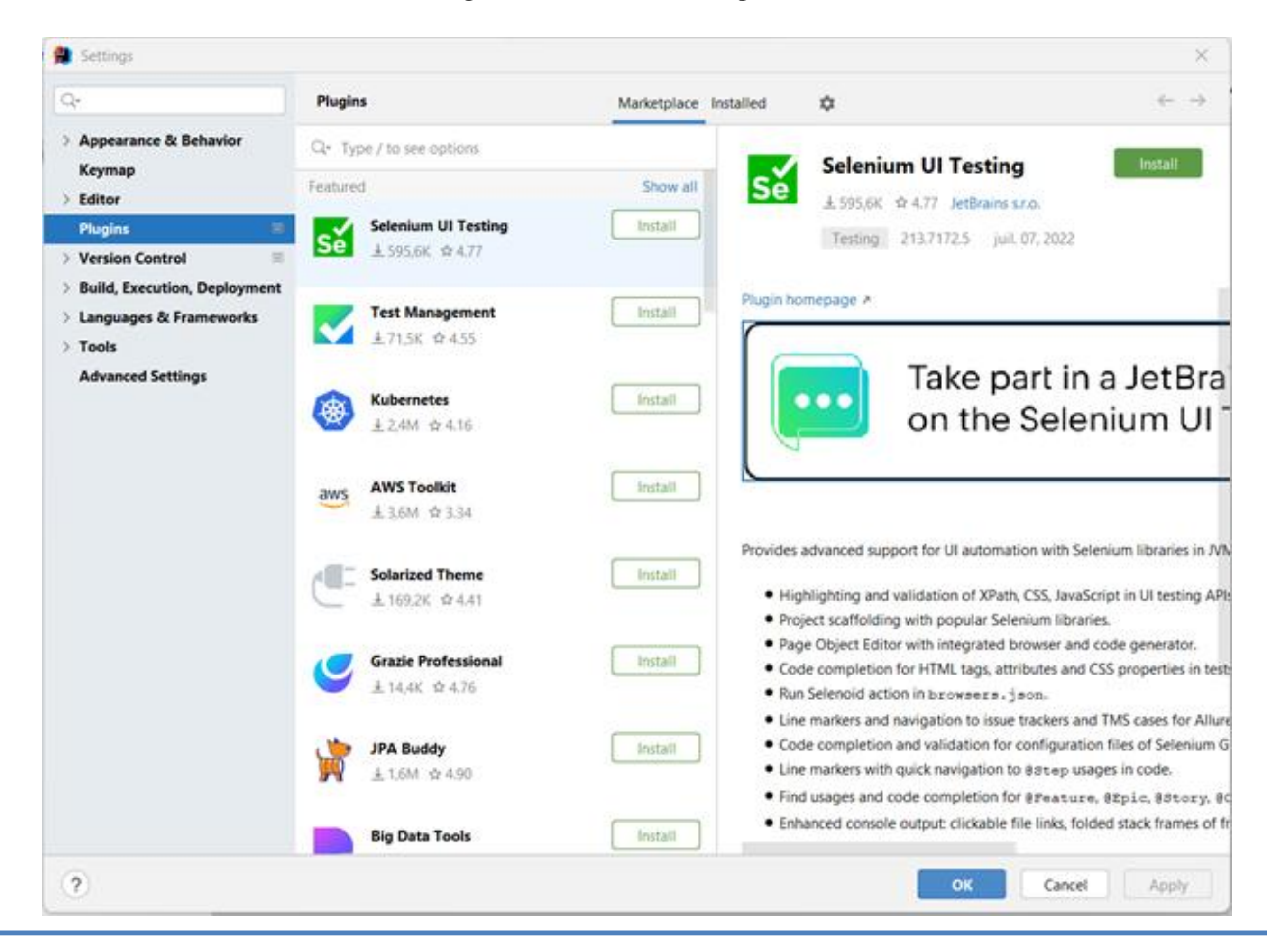
- Pour utiliser le projet lombok avec Intellij, vérifier que le plugin est déjà installé (si le plugin est déjà installé, alors aucune installation n'est nécessaire) :

  ✔ Aller vers File Settings Plugins **Installed**



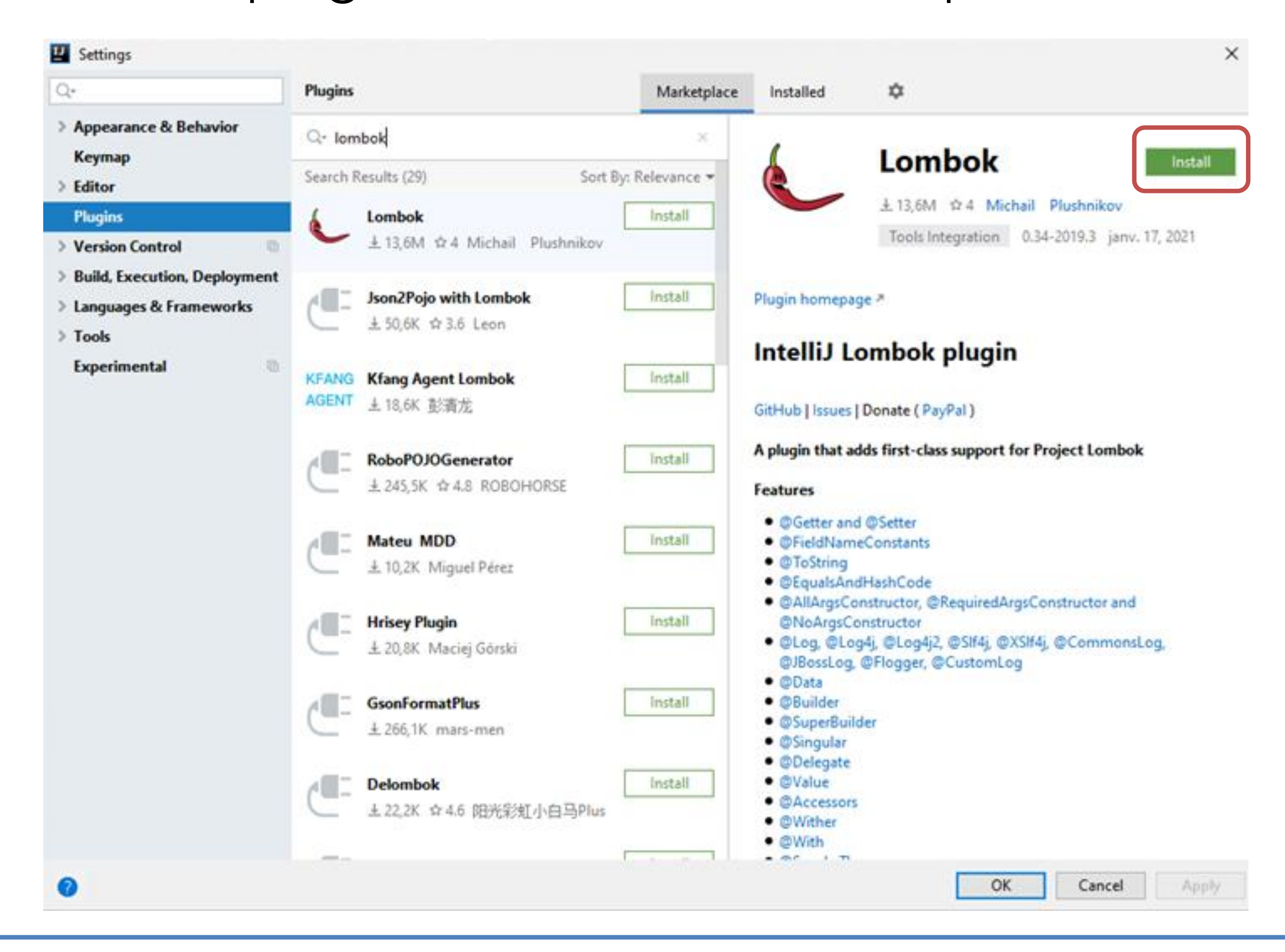
INSTALLATION - PLUGIN

- **Si le plugin n’est pas installé**, alors il faut l’installer :
  ✔ Aller vers File Settings Plugins **MarketPlace**



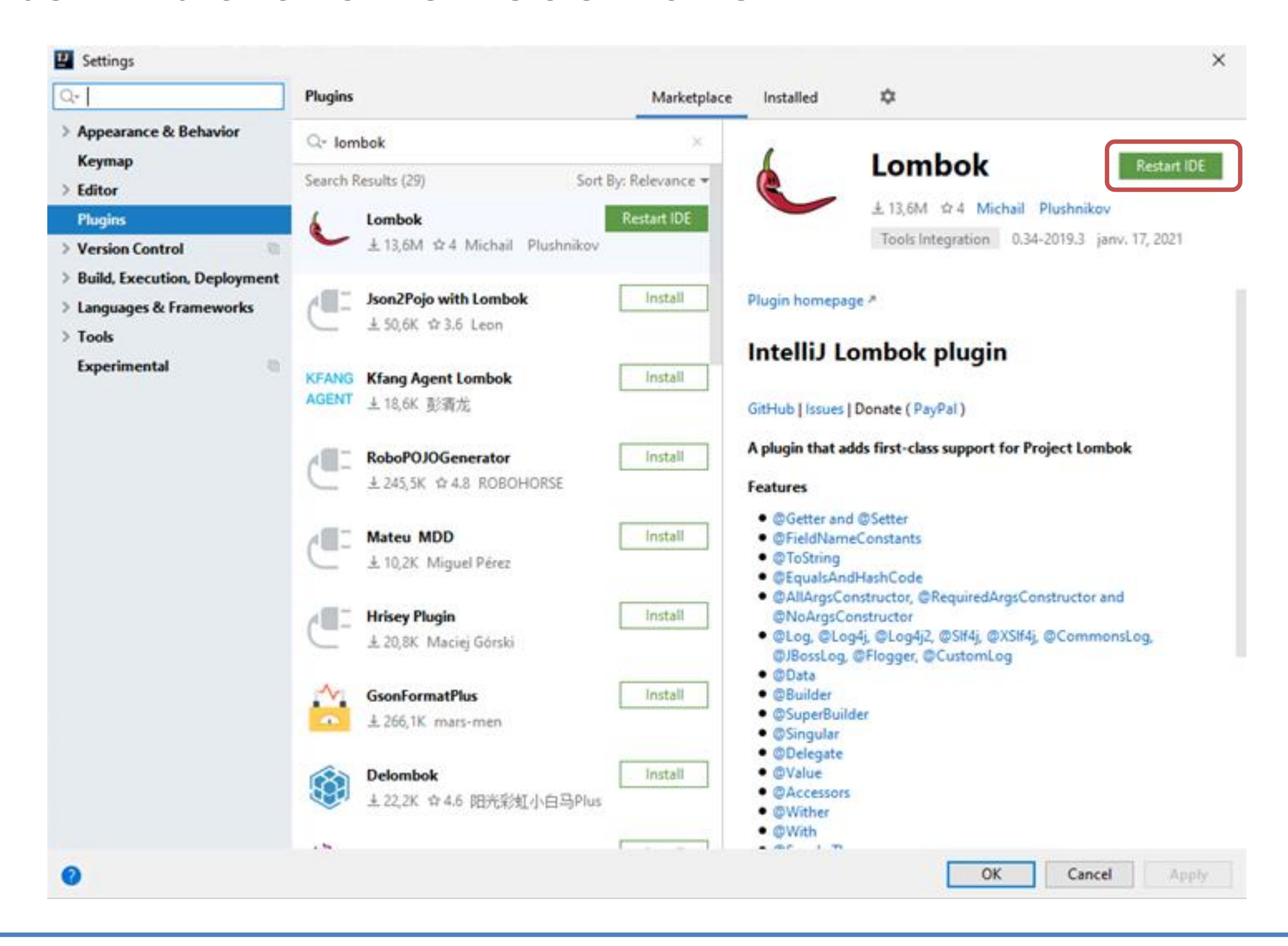
INSTALLATION - PLUGIN

✔Chercher le plugin de « lombok » et cliquer sur « install »



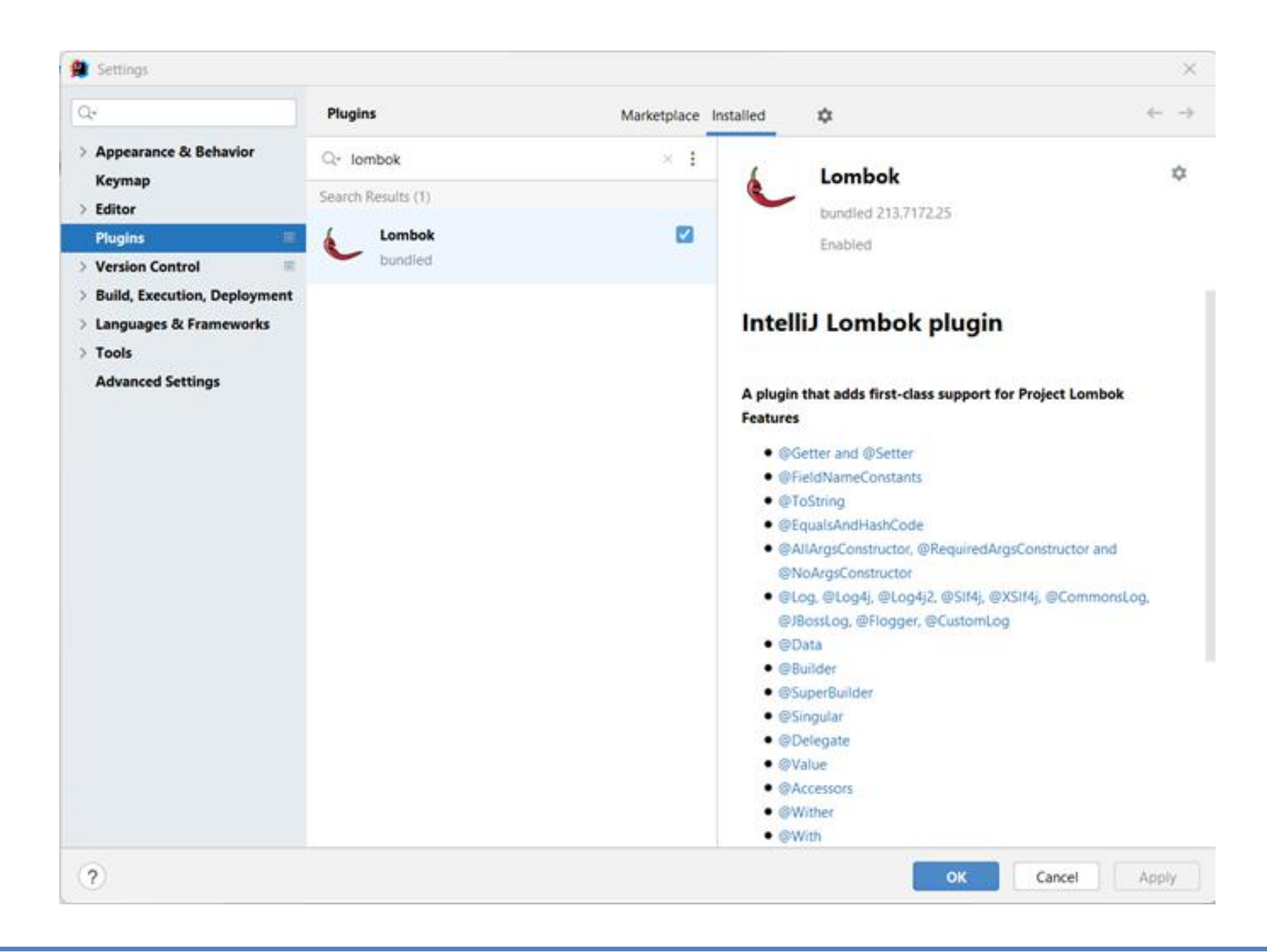
INSTALLATION - PLUGIN

✔ Finaliser l'installation et redémarrer l'IDE



INSTALLATION - PLUGIN

✔Résultat final



LES ANNOTATIONS DU PROJET LOMBOK - GETTERS ET SETTERS

- En ajoutant les annotations **@Getter** et **@Setter**, nous avons demandé à Lombok de générer les getters et les setters pour tous les attributs de la classe.

```java
@Entity
@Getter
@Setter
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

}
```

```java
@Entity
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }
```

LES ANNOTATIONS DU PROJET LOMBOK - CONSTRUCTEURS

- **@NoArgsConstructor** permet la génération de constructeur par défaut.

```java
@Entity
@Getter
@Setter
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

    public Employe() {
        super();
    }
}
```

```java
@Entity
@Getter
@Setter
@NoArgsConstructor
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

}
```

LES ANNOTATIONS DU PROJET LOMBOK - CONSTRUCTEURS

- **@AllArgsConstructor** généré un constructeur avec tous les attributs.

```java
@Entity
@Getter
@Setter
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

    public Employe(int id, String prenom, String nom) {
        super();
        this.id = id;
        this.prenom = prenom;
        this.nom = nom;
    }

}
```

```java
@Entity
@Getter
@Setter
@AllArgsConstructor
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

}
```

LES ANNOTATIONS DU PROJET LOMBOK - CONSTRUCTEURS

- **@RequiredArgsConstructor** définit un constructeur avec seulement les attributs non nuls (@NonNull).

```java
@Entity
@Getter
@Setter
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;
    @NotNull
    private String prenom;
    @NotNull
    private String nom;

    public Employe(String prenom, String nom) {
        super();
        this.prenom = prenom;
        this.nom = nom;
    }

}
```

```java
@Entity
@Getter
@Setter
@RequiredArgsConstructor
public class Employe implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    @NotNull
    private String prenom;

    @NotNull
    private String nom;
}
```

LES ANNOTATIONS DU PROJET LOMBOK - BUILDER

- **@Builder** peut être utilisé pour définir les constructeurs paramétrés.

```
@Entity
@Getter
@Setter
@Builder
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String prenom;
    private String nom;
}
```

```
Employe e = Employe.builder().                .build();
```

```
build() : Employe - EmployeBuilder
equals(Object obj) : boolean - Object
getClass() : Class<?> - Object
hashCode() : int - Object
id(int id) : EmployeBuilder - EmployeBuilder
nom(String nom) : EmployeBuilder - EmployeBuilder
prenom(String prenom) : EmployeBuilder - EmployeBuilder
toString() : String - EmployeBuilder
notify() : void - Object
notifyAll() : void - Object
wait() : void - Object
wait(long timeoutMillis) : void - Object
Press 'Ctrl+Space' to show Template Proposals
```

LES ANNOTATIONS DU PROJET LOMBOK - BUILDER

```java
@Entity
@Getter
@Setter
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

    public Employe(int id) {
        this.id=id;
    }

    public Employe(int id,String nom) {
        this.id=id;
        this.nom=nom;
    }

    public Employe(String nom,String prenom) {
        this.nom=nom;
        this.prenom=prenom;
    }
}
```

```java
Employe e1 = Employe.builder().id(2).build();
```

```java
Employe e2 = Employe.builder().id(2).nom("Ben Foulen").build();
```

```java
Employe e3 = Employe.builder().nom("Ben Foulen").prenom("Foulen").build();
```

- **PS:** L'utilisation de l'annotation @Builder avec l'annotation @RequiredArgsConstructor, peut engendrer des problèmes. Donc, pour définir les constructeurs, il faut utiliser une seule méthode.

LES ANNOTATIONS DU PROJET LOMBOK – TO STRING

- **@ToString** génère une implémentation pour la méthode toString() par défaut. Elle imprimera le nom de la classe, ainsi que chaque champ, dans l'ordre, séparé par des virgules. Par défaut, les champs non statiques sont exclus de toString() généré.

```java
@Entity
@Getter
@Setter
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

    @Override
    public String toString() {
        return "Employe [id=" + id + ", prenom=" +
                prenom + ", nom=" + nom + "]";
    }

}
```

```java
@Entity
@Getter
@Setter
@ToString
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

}
```

LES ANNOTATIONS DU PROJET LOMBOK - EQUALS ET HASHCODE

- Lorsque nous déclarons une classe avec **@EqualsAndHashCode**, Lombok génère des implémentations pour les méthodes **equals** et **hashCode**.

```java
@Entity
@Getter
@Setter
@EqualsAndHashCode
public class Employe implements Serializable {

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String prenom;

    private String nom;

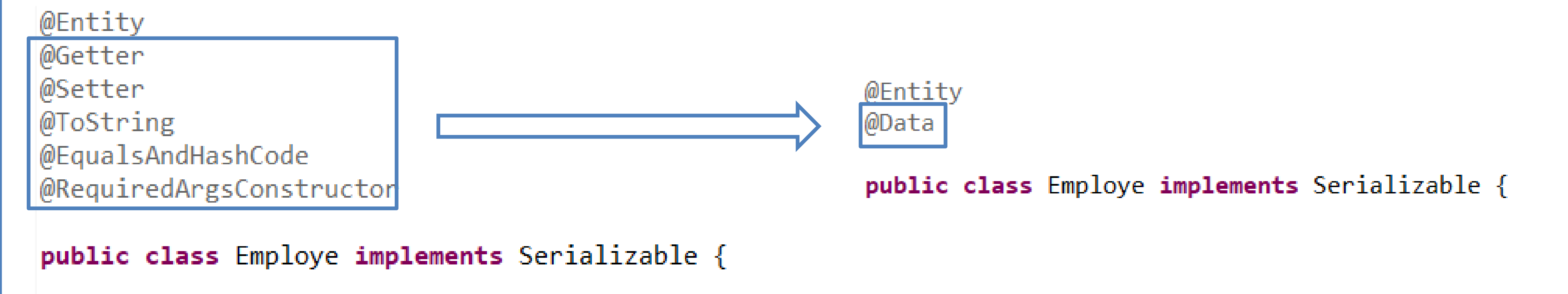
}
```

```java
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + id;
    result = prime * result + ((nom == null) ? 0 : nom.hashCode());
    result = prime * result + ((prenom == null) ? 0 : prenom.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Employe other = (Employe) obj;
    if (id != other.id)
        return false;
    if (nom == null) {
        if (other.nom != null)
            return false;
    } else if (!nom.equals(other.nom))
        return false;
    if (prenom == null) {
        if (other.prenom != null)
            return false;
    } else if (!prenom.equals(other.prenom))
        return false;
    return true;
}
```

LES ANNOTATIONS DU PROJET LOMBOK - @DATA

- **@Data** est l'annotation qui regroupe @Getter, @Setter, @ToString, @EqualsAndHashCode et @RequiredArgsConstructor.



PS : L'annotation **@Data** peut causer des problèmes lors de l'association des entités (manyToMany , OneToMany, etc…) engendrant une gestion assez compliquée de ces sessions (utilisation du terme exclude , etc…)

Pour votre projet, privilégier l'utilisation des annotations @Getter, @Setter, … pour une utilisation plus simple de Lombok.

LES ANNOTATIONS DU PROJET LOMBOK - MODIFICATEURS D'ACCÈS

- Pour modifier l'accès à l'attribut, on ajoute l'annotation suivante:

@FieldDefaults(level = <Format>)

```java
@Entity
@FieldDefaults(level = AccessLevel.PRIVATE)
public class Employe implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int id;
    String prenom;
    String nom;
}
```

```java
@Entity
@FieldDefaults(level = AccessLevel.PROTECTED)
public class Employe implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int id;
    String prenom;
    String nom;
}
```

```java
@Entity
@FieldDefaults(level = AccessLevel.PUBLIC)
public class Employe implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    int id;
    String prenom;
    String nom;
}
```

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER

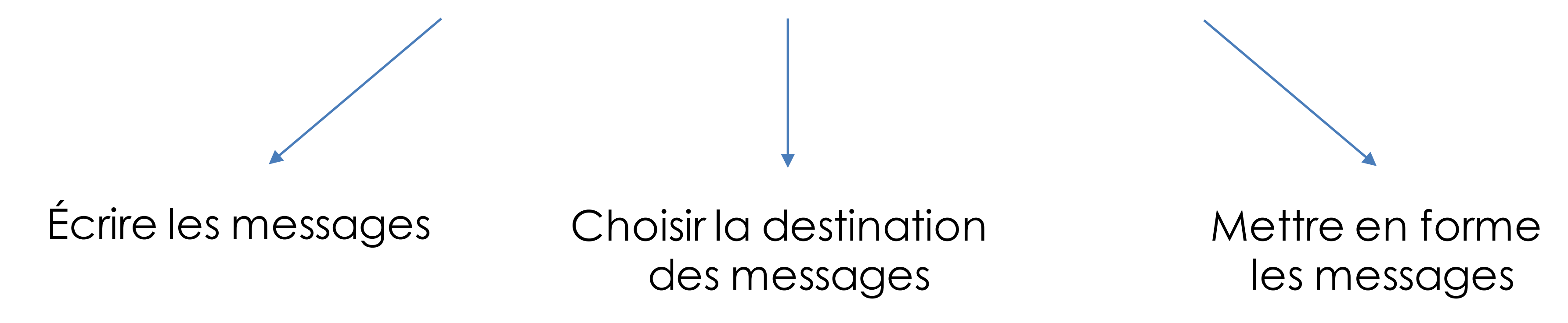
Le LOGGER sera vu en séance 10 et non en séances 3

(Séance 3 : nous avons utilisé les accesseurs (getters setters), les constructeurs, les modificateurs (private) de Lombok, pour créer nos premières entités).

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER

- Les bibliothèques de journalisation met trois sortes de composants à disposition du programmeur :

les **loggers**, les **appenders** et les **layouts**.



LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : COMPOSITION

- L'**appender** désigne un flux qui représente le fichier de log et se charge de l'envoie de message formaté à ce flux (Console ou fichier de log).
- Un **logger** peut posséder plusieurs **appenders**. Si le logger décide de traiter la demande de message, le message est envoyé à chacun des appenders.
- Le **layout** permet de définir le format du message de log. Il admet un format par défaut fourni par la bibliothèque.

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : NIVEAUX DE LOG

- Le logging gère des priorités, ou Level, pour permettre au logger de déterminer si le message sera envoyé dans le fichier de log (ou la console). Il existe six priorités qui possèdent un ordre hiérarchique croissant :

TRACE ⇨ DEBUG ⇨ INFO ⇨ WARN ⇨ ERROR ⇨ FATAL

	FATAL	ERROR	WARN	INFO	DEBUG	TRACE
FATAL	x					
ERROR	x	x				
WARN	x	x	x			
INFO	x	x	x	x		
DEBUG	x	x	x	x	x	
TRACE	x	x	x	x	x	x
ALL	x	x	x	x	x	x

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

- En utilisant l'annotation **@SLF4J** (Simple Logging Facade for Java):
  - On génère un logger de la classe annotée sans la déclaration de la constante.
  - L'annotation fournit une variable statique appelée « log » qui fournit les utilitaires de journalisation par défaut

```java
import org.springframework.stereotype.Service;
import lombok.extern.slf4j.Slf4j;

@Slf4j
@Service
public class DemoService {
	//...
}
```

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

```java
import lombok.extern.slf4j.Slf4j;
@Service
@Slf4j
public class ClientServiceImpl implements IClientService {

    @Autowired
    ClientRepository clientRepository;


    @Override
    public List<Client> retrieveAllClients() {
        List<Client> clients = (List<Client>) clientRepository.findAll();
        for (Client client : clients) {
            log.info(" client : " + client);
        }
        return clients;
    }
```

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

- Pour spécifier le fichier externe où les messages sont stockés dans un projet Spring Boot, il suffit d'ajouter cette instruction dans le fichier **application.properties** de l'application:

```
logging.file.name= <path>
```

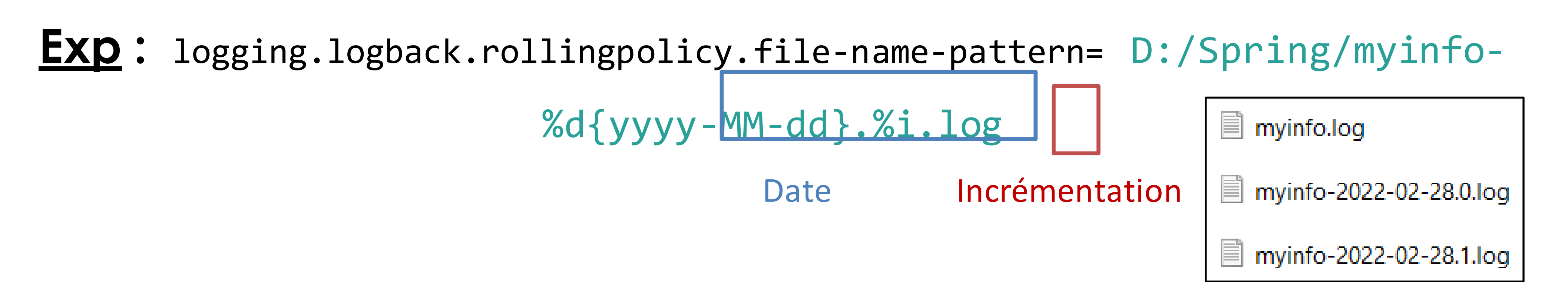
<u>Exp</u> : `logging.file.name=D:/myinfo.log`

myinfo.log

- Pour personnaliser le nom du fichier, il suffit d'ajouter cette propriété dans le fichier **application.properties** de l'application:

```
logging.logback.rollingpolicy.file-name-pattern= <Pattern>
```

<u>Exp</u> : `logging.logback.rollingpolicy.file-name-pattern= D:/Spring/myinfo-%d{yyyy-MM-dd}.%i.log`



LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

- Pour spécifier un seuil pour la taille du fichier externe, il suffit d'ajouter cette instruction dans le fichier **application.properties** de l'application :

```
logging.logback.rollingpolicy.max-file-size= <taille>
```

Exp : `logging.logback.rollingpolicy.max-file-size= 10KB`

Cela maintient la taille maximale de chaque fichier externe autour de 10 Ko. Un nouveau fichier sera créé si la taille du fichier actuel dépasse ce nombre.

 Si cette propriété n'est pas spécifiée, la taille par défaut est de 10 Mo.

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

- Pour personnaliser le message de log, il suffit d'ajouter cette instruction dans le fichier **application.properties**
  - ✔ Dans la console: `logging.pattern.console= <Format>`
  - ✔ Dans le fichier: `logging.pattern.file= <Format>`

Exemple:

```
logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} - %-5level- %logger{36} - %msg %n
```

Motif	Rôle
%C	Le nom de la classe qui a émis le message
%d	Le timestamp de l'émission du message
%m ou %msg	Le message
%n	Un retour chariot
%level	Le niveau de log (info,debug, fatal ,etc..)
%logger	Des informations sur l'origine du message dans le code source (package, classe, méthode)

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

- Les niveaux de journalisation peuvent être définis dans le projet Spring Boot en définissant ses configurations dans le fichier application.properties.
- Le format pour définir la configuration du niveau de journalisation est :

```
logging.level.[classpath] = [level]
```

Exemple : `logging.level.com.service.DemoService= WARN`

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE

- Pour spécifier un niveau de journalisation pour toutes les classes qui n'ont pas leurs propres paramètres de niveau de journalisation, le root logger peut être défini à l'aide de:

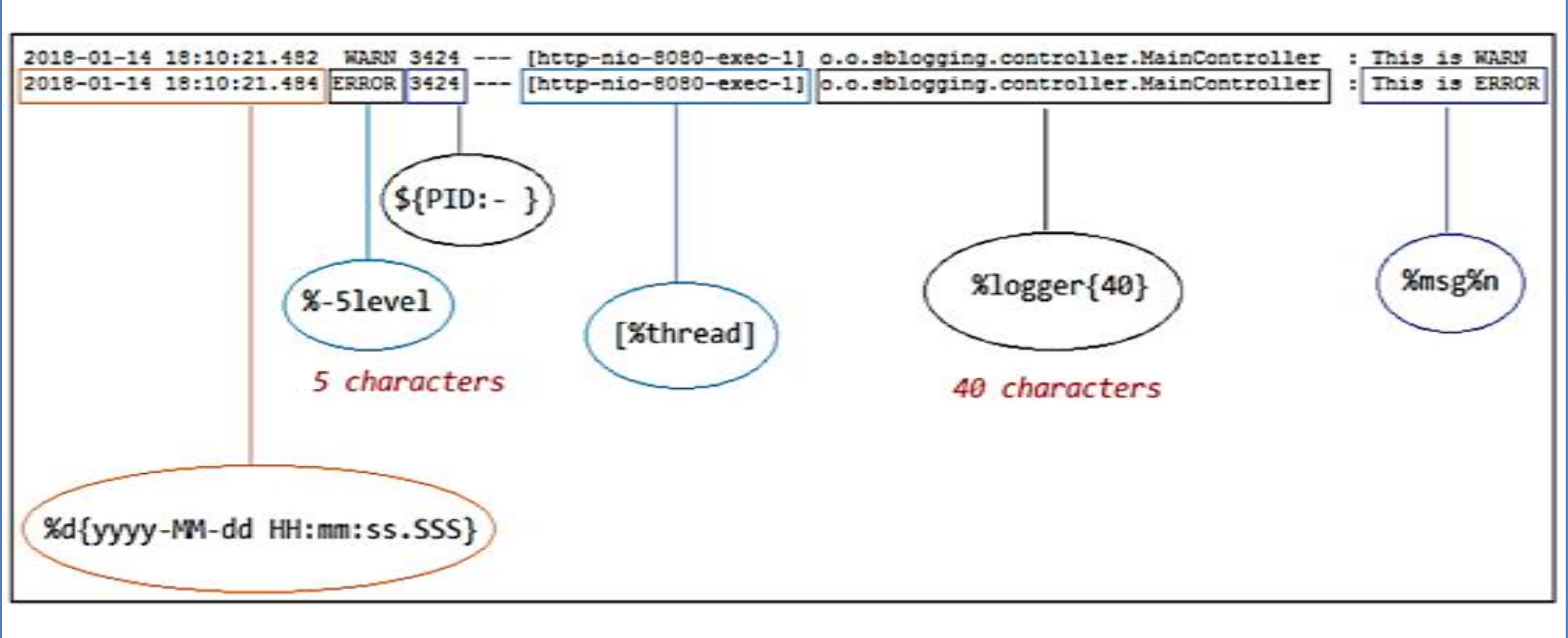
```
logging.level.root = [level]
```

Exemple : `logging.level.root= DEBUG`

PLACE A LA PRATIQUE

- Appliquer les annotations de ce cours sur le projet de l'étude de cas pour générer:
  - ✔ Les getters
  - ✔ Les setters
  - ✔ Les constructeurs
  - ✔ La méthode « ToString »
  - ✔ La journalisation

LES ANNOTATIONS DU PROJET LOMBOK – LOGGER : MISE EN OEUVRE



Si vous avez des questions, n’hésitez pas à nous contacter :

Département Informatique
UP ASI
Bureau E204